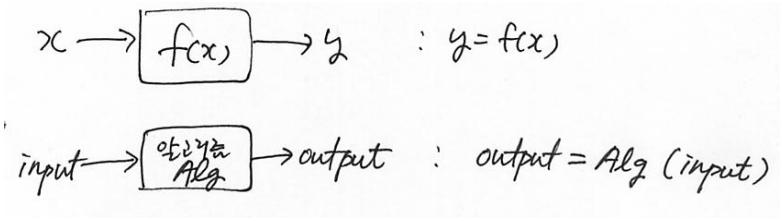
6. 함수적 세상

2010 데이터로 표현하는 세상 요약본
고려대학교 김현철 교수
hkim64@gmail.com

6.1 Function 함수

알고리즘은 일종의 함수(function)이라고 이야기 할 수 있다. 즉, 어떠한 input값을 output값으로 바꾸어 주는 함수와 같다. 우리가 알고 있는 수학의 함수와 비교하여 보자.



똑같지 않은가?

다만 차이점은, f(x)는 수학적 공식만 사용하고, x와 y는 모두 수치 값만 사용하지만, 알고리즘은 수학적 공식뿐만 아니라 데이터를 처리하는 논리적인 절차로 표현하고 input과 output값은 수치 값뿐만 아니라 우리가 이 세상에서 접하게 되는 모든 정보를 다 다룰 수 있다.
알고리즘이 수학적 함수보다 훨씬 더 강력한 것이라고 할 수 있다.

이제, 우리가 그 동안 공부했던 모든 알고리즘은 함수(function)의 관점에서 다시 정의를 해야 한다. 즉, 만약에 알고리즘의 이름을 Reverse라고 한다면 그리고 reverse는 문자배열 A와 배열의 크기 n을 input으로 받고 또 다른 문자배열 B를 output으로 낸다고 한다면 다음과 같이 표현할 수 있다.
B = Reverse(A, n);

이것을 알고리즘에서도 "함수 Reverse" 혹은 "Reverse function" 이라고 부른다. (function 이라는 용어 대신에 procedure 혹은 subroutine이라는 용어를 사용하기도 한다)
함수 Reverse는 다음과 같이 정의된다. (마치 수학함수에서 f(x)=x*x 로 정의 하는 것 처럼)

```
Reverse(A, n) {
      ….
      ….
      Return B;
   }
```

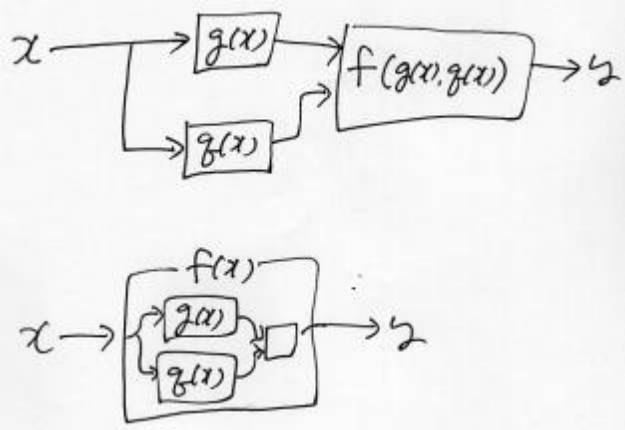
수학함수에서 숫자를 제곱하는 것도 알고리즘으로 정의하면

```
squared(x) {
   y=x*x;
   return y;
   }
```

로 정의하면 된다.
또는 더 간단하게 표현한다면

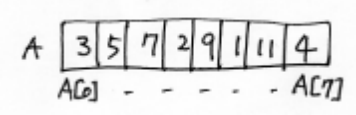
```
squared(x) {
    return x*x;
    }
```
로 정의해도 된다.
그러면 다른 알고리즘(즉 다른 함수)에서 그 함수를 다음과 같이 사용할 수 있다.
```
double_squared(x) {
    y=2*squared(x);
    return y;
}
```

이것은 마치 수학함수에서 다른 함수를 이 함수의 domain값으로 사용하는 것과 같다. 즉, x에 대하여 f(x)와 g(x), q(x)가 정의되어 있을 때, y=f(g(x)+q(x)) 처럼 쓸 수 있는 것과 같다.



Example)
다음과 같은 숫자가 있다고 할 때, 그 집합 안에 있는 모든 숫자의 합을 구하는 알고리즘을 작성해보자.

{ 3, 5, 7, 2, 9, 1, 11, 4 }
자, 더하려고 하는 숫자들의 집합을 하나의 array(행렬)로 표현하여 보자.



그러면 합을 구하는 알고리즘을 함수(function)의 형식으로 표현한다면
```
summation(A) {
……
return total_sum;
}
```
이 될 것이다.
반복하여 A[i]에 있는 값들을 total_sum에 더하여서 그 최종값을 return하면 된다.

```
Summation(A) {
n= size of array A;
total_sum=0;
for(i=0; i<n; i++) {
total_sum=total_sum+A[i];
}
return total_sum;
}
```
간단!

i가 0부터 시작하여 하나씩 증가하여 n보다 작을 때 까지 { }안의 내용을 반복! 다음과 같은 의미이다.

```
i=0;
while (i<n) {
…
i = i+1;
}
```

마찬가지로 하여, 집합A에 들어 있는 값들의 전체 곱을 구하는 알고리즘을 만들어 보자.

```
Product(A) {
n= size of array A;
total_prod=1;
for(i=0; i<n; i++) {
total_prod=total_prod*A[i];
}
return total_prod;
}
```

간단!

그러면 만약에 우리가 summation과 product라는 함수를 이미 알고 있다고 가정을 할 때, 새로운 함수 proddivsum(A)을 정의한다고 하자. 이 함수는 집합 A에 들어 있는 수치값들의 전체 곱을 전체합으로 나눈값을 return하는 것으로 해보자.

```
Proddivsum(A) {
value=product(A)/summation(A);
return value;
}
```

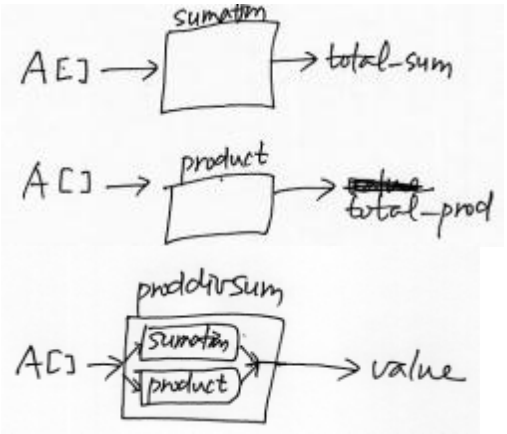
너무나 간단!

하지만, 조금 더 생각해보면 나눗셈에서 분모가 0이 되면 안 된다는 것을 우리는 알고 있다. 그래서 위의 함수를 다시 정의하여 보면, 0으로 나누는 것을 제외 시켜야 한다.

```
Proddivsum(A) {
If (summation(A) != 0)       ( != 는 not equal 이라는 표시 )
Then { value=product(A)/summation(A);
return value;
}
Else return error;
}
```

라고 하면 정확하다.

함수의 그림으로 다시 표현하여 본다면,



이 세상은 함수의 조합으로 표현할 수 있다. 이 세상은 하나의 함수이다! 〈김현철 2004〉

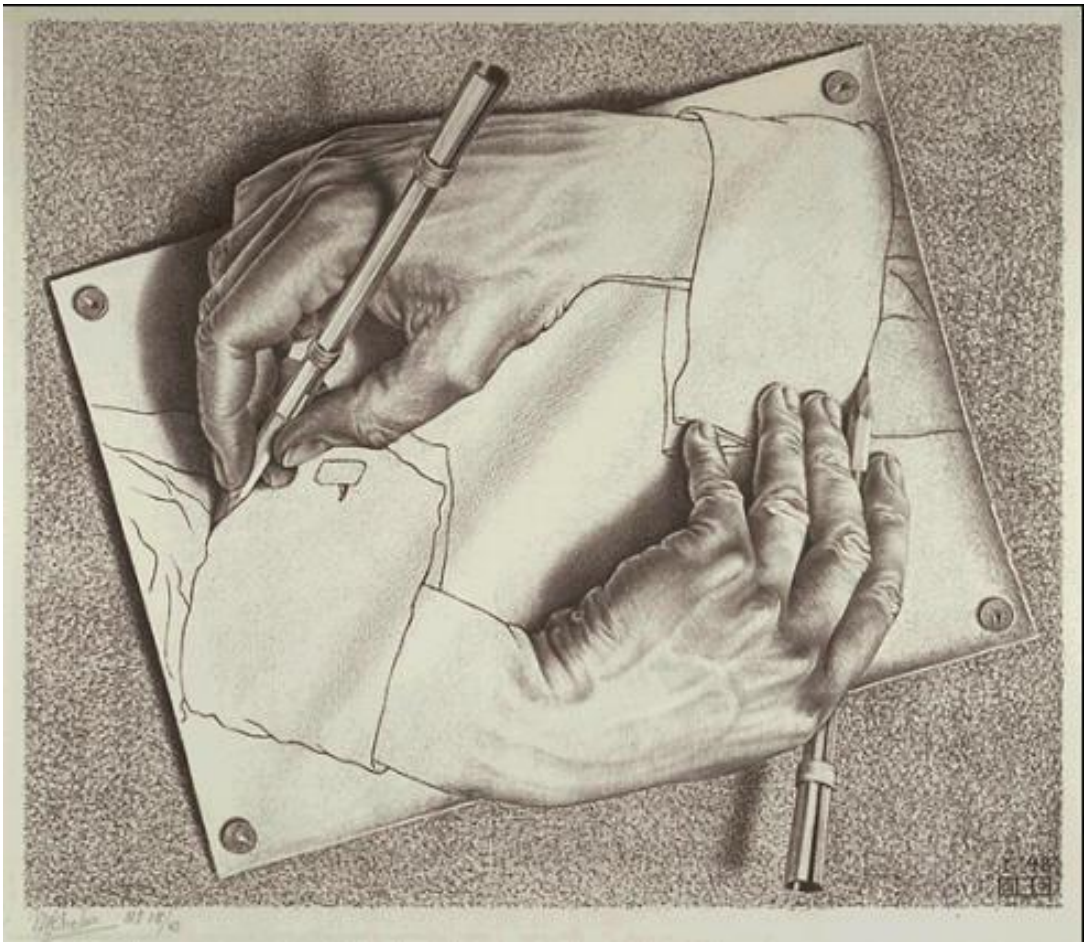
```
Life(man) {
Eat(food);
Love(her);
Sleep(8);
}
```

각각의 함수를 또 다시 정의할 수 있다.

알고리즘이나 절차는 어떠한 특정한 함수(function)에 보관되며, 누군가 그 함수를 호출(call)하면 함수 내부에 저장되어 있는 알고리즘이 실행된다. 위와 같이 squared 함수 안의 알고리즘은 매우 짧고 간단하지만 실제적으로는 매우 복잡하고 정교한 경우가 많다. 따라서 필요한 함수를 잘 정의한 다음, 그들을 적절히 이용하여 알맞은 위치에 불러서 쓴다면 복잡한 논리도 뒤엉켜 있는 알고리즘을 간결하고 아름다운 공식으로 만드는데 아주 중요한 역할을 담당한다.
그런데, 함수에서 재미있는 것은 함수가 내부에서 자기 자신을 부르는 것도 가능하다는 사실이다.

6.2 Recursive World (재귀적 세상)

네덜란드의 화가 Maurits Cornelius Esher의 작품은 절묘하게 뒤얽혀 있는 시각적인 패턴이 불러 일으키는 신비로운 유혹과 기묘하게 뒤틀린 물리적 공간의 낯선 분위기로 사람들을 유혹한다.



Drawing Hands, 1948
출처 : http://en.wikipedia.org/wiki/M._C._Escher

이러한 Esher의 작품세계는 환상적인 세계를 그리고 있는 것 같지만, 사실은 우리 자신이 살아가고 있는 세계에 숨어 있는 시각적이고 공간적인 법칙을 나타내고 있으므로 단순한 환상이라고 보기는 어렵다.

Recursive function의 예를 보도록 하자.
n의 Factorial값은 다음과 같이 정의할 수 있다. n*(n-1)*(n-2)*…*2*1
어떤 input 값 n을 받아서 n의 factorial 값을 출력하는 함수 Factorial(n)을 정의하여 보시오.
우리가 알고 있는 보통 looping을 이용하여 pseudo code를 만들면

```
factorial(n) {
  total=1;
  for( i=1; i<=n; i=i+1) {
    total=total*i;
    }
  return total;
  }
```

이것을 recursive하게 정의하기 위하여서는 factorial을 다시 한번 살펴보자. factorial(n)은 n*factorial(n-1)로 표현할 수 있다.
따라서

```
factorial(n) {
  return n*factorial(n-1);
  }
```

로 표현하면 된다. 맞는가?
문제가 있다. 언제까지 계속 recursive하게 갈 것인가. 언제 멈출 것인가?
We need "termination condition"! 언제 멈춰야 하는지에 대한 조건이 필요하다.

```
factorial(n) {
  if(n==1) return 1;
  return n*factorial(n-1);
  }
```

이 되어야 한다!

피보나치(Fibonacci) 수열
피보나치 수열은 두 개의 1로 시작하여 앞의 두수를 더한 수가 현재의 수가 되는 방식으로 진행한다. 예를 들어
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …
피보나치 수열을 수학공식으로 나타내면 다음과 같다.
F(n) = 1, where n <= 2
F(n) = F(n-1)+F(n-2), where n>2

이 피보나치의 수열을 계산해 내는 알고리즘 함수 Fib(n)을 생각해보자. 즉, n을 입력하면 Fib(n)은 Fibonacci 수열에서 n번째 숫자를 출력한다.

```
Fib(n) {
  If (n==1) then return 1;
  If (n==2) then return 1;
  temp1=temp2=1;
  for(i=3; i<=n; i++) {
    current=tem1+tem2;
    temp1=temp2;
    temp2=current;
    }
  return current;
  }
```

이것을 recursive function으로 표현한다면

```
Fib(n) {
  If (n<=2)    ➔ 종료조건
    then return 1;
    else return Fib(n-1) + Fib(n-2);
  }
```

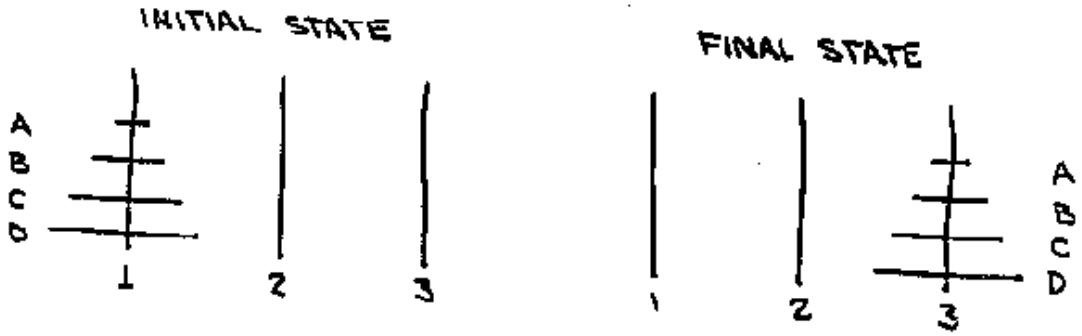
이렇게 재귀적인 함수를 이용하여 문제를 해결하는 것은 많은 경우에 문제를 간단하게 해결할 수 있는 실마리를 준다.

Hanoi Tower

There are 4 disks(A,B,C,D) of graduated sizes initially stacked on peg1 of three pegs with A, the smallest, on top and D, the largest, at the bottom. The disks are to be transferred to peg 3 observing the following rules:

(1) Only one disk can be moved at a time &

(2) No disk can be placed on top of a smaller disk.



일반화된 방법으로 설명한다면,

Given 3 peg i, j and k, the problem of moving a stack of size n>1 from peg I to peg k can be replaced by the three problems:

(1) move n-1 disks from i to j

(2) move 1 disk from i to k

(3) move n-1 disks from j to k

recursive하게 이 문제를 해결하여 보시오.

Ex) n=4일 때,

.끝.